Event Visualization in High Energy Physics

Howard Stone *Princeton University, Princeton, USA*

Abstract

The role of event visualization software in the overall context of a high energy physics experiment is reviewed with emphasis on its integration into the overall software structure of the experiment. The subdivision of the graphics tasks into "local" and "global" tasks and the related issue of interactive control of these tasks is discussed. The importance of this distinction in the early phases of development of future LHC visualization systems is emphasized.

Keywords: High Energy Physics, Event Visualization, Interactivity, Graphics

1 Introduction

Data corresponding to an electron-positron annihilation event at LEP and recorded in one of the LEP detectors typically consists of about 2000 measurements of position and/or deposited energy and/or timing. After passing through the experiment's reconstruction algorithims and being combined with calibration constants and other data from the experiment's database, these measurements are reduced to about 200 higher level reconstructed objects such as tracks, electromagnetic clusters, hadronic clusters, etc. The use of computer graphics in visualizing these data is essential for conveying the underlying physics of the "events" to a physicist.
Computer graphics is especially useful when the information being portrayed is positional in nature and spatial correlations are important. Color and other visual cues can be employed to convey additional information to the physicist concerning the amount of energy deposited or timing information at the spatial points represented in the computer display. A visual representation of the data as an "event display" showing the correspondence between the data and the detector geometry provides the physicist with an intuitive feel for what has occurred in the detector and can be understood by the non-specialist. Event displays are, therefore, a valuable educational and public relations tool[1].

2 Present day Event Visualization Software in High Energy Physics

Todays event display, or "scan", programs may be characterised as follows (for reviews of software associated with particular LEP experiments see [2]):

> A collection of software modules capable of *displaying all data* from a HEP experiment and providing *real-time interactive* control of the display, *processing and I/O* of the data.

The data is typically displayed as three dimensional images of tracks, other reconstructed objects and their corresponding raw detector information, drawn together with a three dimensional representation of the detector's geometrical layout. Two dimensional representations are also employed and generally experiments like to provide options for running the software on high-end (with hardware graphics support) as well as low-end graphics workstations. Two dimensional representations of the data are also usefully employed to display "fish-eye" or other distorted views of the event so that a range of sub-detectors of vastly different sizes can be included in the same image [3].
Data is also be displayed as text in the drawing area (e.g. timing information written next to a scintillator hit) or as text in a special text area (e.g. a list of all the electromagnetic clusters in the event together with their energies) - such text areas are commonly pop-up pop-down widgets so as to conserve valuable screen space. Extensive cross referencing between all the displayed textual data is crucial for conveying the maximium amount of information. Tracks drawn and numbered in the display should correspond to tracks in numbered lists so that momenta, hit information and other non-graphical information can be accessed immediately. Specialized displays are also incorporated

in the graphics areas for displaying plots of the data associated with reconstructed objects, for example, residuals of the hits associated with reconstructed tracks. In some cases high level interactive commands are provided that permit the user to direct the reconstruction - for example, individual hits may be removed from a particular track.
Visualization software running on high-end workstations typically employs a “display list” of some kind. This is an area of memory in the workstation in which the information to be displayed is stored. Once this information has been stored it maybe displayed in various ways, eg, rotated, shaded, depth cued, etc, by functions implemented in hardware that are transparent to the rest of the software and very fast.
By exploiting visibility masks, the degree of detail presented in the drawing area can be controlled. Since visibility masks are processed when the display list is traversed, items can be switched on or off very rapidly. This control is an example of “local interactivity”. A crucial question to be addressed at the design stage of the software is just how much information to write into the display list automatically and how much requires that the user specifically requests that the information be written.
In no particular order of importance, present day software has, with varying degrees of success attempted to meet the following general design goals:

- When the software is running in the online environment it should display the raw hits in the detector, when being used as a debug tool on the reconstruction code it should be capable of displaying reconstructed objects and when being used as an analysis tool it should provide representations of physics objects (maybe as simple as four-vectors).
- Whenever the software is running it should have access to any of the data that was used to construct the objects being viewed. The program should also have access to and be capable of displaying the calibration, and other, data currently loaded by the software.
- The display software can therefore provide users with a transparent interactive means of accessing the experiment’s database and may eventually evolve into a unified physics analysis system.

A useful display program will inevitably evolve into a system with many functions and possible uses. It is essential that the user be able to direct the display of information, processing and analysis of the data, and the i/o streams being accessed by the software interactively. A large fraction of the code will, therefore, be devoted to the user interface.

3 The Importance of Event Visualization in HEP

The experiment’s visualization software evolves as the experiment’s needs change. Initially it is used for optimizing and debugging the detector layout and geometry and for performance studies of the detector using simulated data. It is also a useful tool for developing the various reconstruction algorithms. When the detector is being commissioned it plays a vital role in the verification of the detector and reconstruction software performance and often reveals subtle software bugs. As the experiment stabalizes and physics analyses progress it is used for general monitoring, checking acceptances and backgrounds in selected physics events, as a tool for detailed study of events of special interest, as a means for physicists to “keep their hands on” the analysis, as a tool for preparing publications and as a public relations utility.
The evolving nature of the software requires that it be as modular as possible to allow code to be re-used and that the experiment treats its devolopment and maintenance as a priority over the course of the experiment.

4 Overall Design and Implementation of Event Display Software Systems

The general flow of data through, and the user interaction with, the visualization software is illustrated in some detail in Figure 1. The functionality of the visualization system falls into four main categories:

1. Accessing the data structures containing the data associated with the event and the geometrical specification of the detector. This may include raw data, reconstructed data, summary data, simulated data etc.
2. The mainipulation of the above data objects into a form suitable for translation into graphics primitives and subsequent presentation as displayed objects. This requires the association of specific graphics tags to the data, pick identifiers, visibility masks, viewport identifiers, color information etc.
3. The presentation and manipulation of the graphics data in the display. This would normally consist of manipulations of elements of a display list in one form or another.
4. Provision of a comprehensive user interface to control the various options and implementations of the above functions.

The reconstruction code combines data from the experiment with calibration constants and geometric information about the detector from the database into reconstructed objects such as clusters in the calorimeters. The graphics software then prepares graphics objects, lists of primitives, which are sent to the display list for rendering on the display. The user may interact “locally” to manipulate the rendering of the contents of the display list, rotating, shading, zooming the image, etc. Once the display list has been prepared, these local tasks are independent of the rest of the software.
In order that the user be able to correlate the physics information being displayed with the physical location of the detector elements the software also has to be provided access to a complete, accurate and up-to-date, specification of the detector geometry. Since the requirements for access to geometry

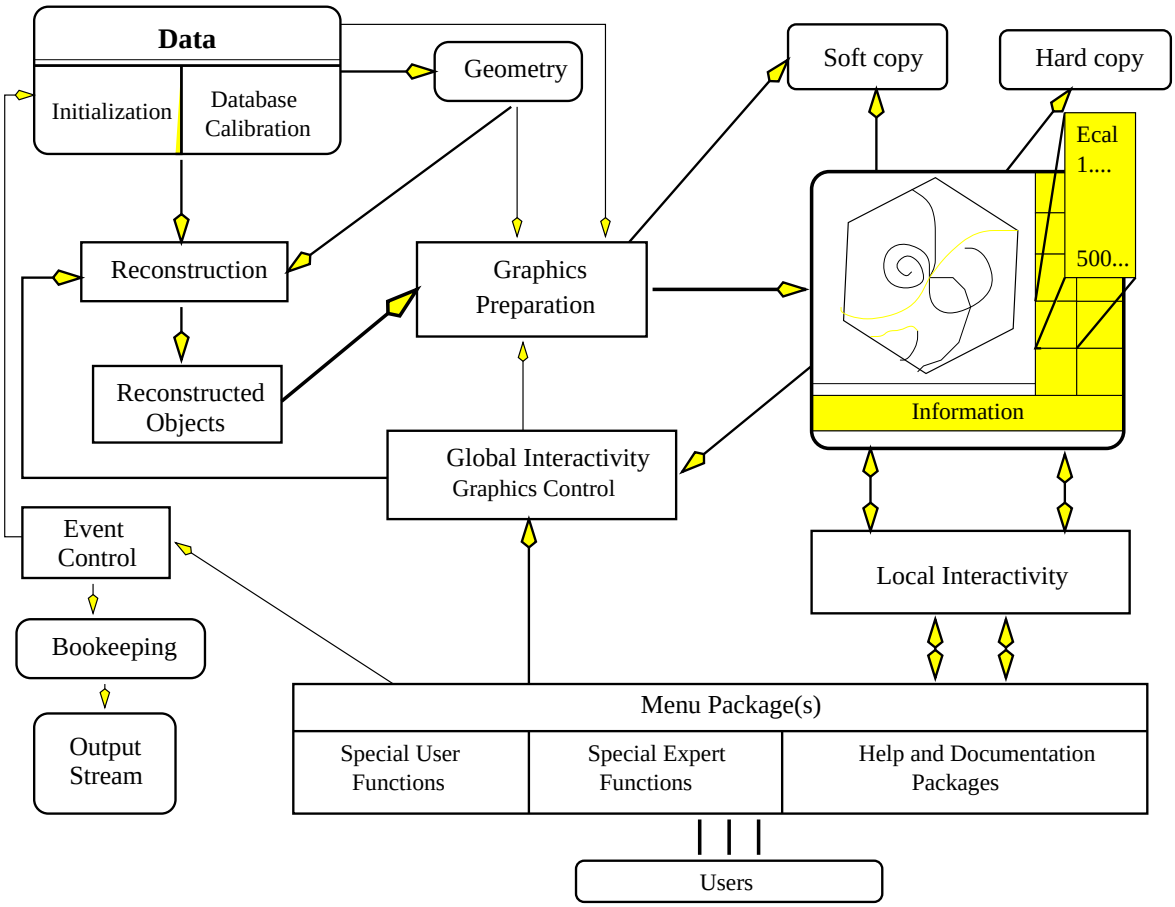

Figure 1: The basic modules of a complete event visualization system, including the links between them. These links represent both the main data flow and the interactive links controling program flow.

are the same in reconstruction/display as in simulation, it is natural to use the same geometrical database and detector specification for both tasks. The visualization software is thus provided with a series of hooks into the Geant framework to allow for representtion of the detector and to vastly simplify the task of visualizing simulated data, even to the extent of providing realtime images of the tracing phases of the simulation itself. For the future, it is essential that display software possess these same hooks into the Geant4 geometrical specfications.
The user also controls the preparation of the display list from the graphics objects, for example he may require that only tracks with a transverse momentum above some cut be written. This is one type of "global interactivity" which requires that the display communicate with the rest of the reconstruction and analysis software. All of the tracks may have been prepared as graphics objects, or this change in the display may require that the entire track reconstruction be re-run. Another type of global interactivity might consist of selecting some points on a track to be excluded from the track fit. The points would be selected (picked) from the display and the program instructed to re-reconstruct the track(s) in question.
The design of the graphics objects requires careful attention. Consider the display of crystals in a calorimeter that the reconstruction has determined form a cluster corresponding to a single particle. The graphics object might consist of the energy in each crystal together with the length and coordinates of the center of the crystal. When the graphics object is transfered to the display list the user can control the representation, the crystal might be drawn with its true geometry and colored according to its energy, or it might be colored according to its association with other objects and drawn with its length scaled according to its energy.

4.1 The Distinction between Local and Global Interactivity

Tasks 1 and 2 above are performed by software modules that must be intimately connected to the main body of the reconstruction code, while tasks 3 and, to some extent, task 4 maybe considered to act only on the display list and are, thus, independent of the bulk of the software. It is proposed to term the functionality associated with 1 and 2 as *global functionality* and the handling of user requests that manipulate these tasks as *global interactivity*. Those tasks that a performed locally on the display list in 3 and 4 we propose to call *local* and their associated user manipulation we refer to as *local interactivity*.
This distinction is important for three main reasons:

- It transparently maps over to a client/server model as illustrated in Figure 2.
- Tasks termed local are (or maybe in the future) typically implemented in hardware (or highly optimized system functions), so that they are perceived to be manipulated instantaneously by the user.
- The client/server model greatly aids the software development task and may provide direct input to the hardware issues associated with the implementation of the experiment's software and computing plan. The server may run on a few "mainframe" type systems with their inherent massive data handling capabilities, while the clients might be running on numerous workstations installed on every active physicist's desk.

The question as to where the dividing line between local and global functionality is to be drawn is central to the implementation of the final visualization system and is *the* key design decision at the outset of the development of the experiment's software - both for event visualization and routine physics analysis tasks. In dealing with this issue the following questions need to be considered:

- Where is the data resident, the geometric specifications of the detector, the calibration constants, etc?
- What software modules have access to and control of the processing of the data being displayed (is the data already in the display list)?
- What are the cost/performance issues associated with the client/server data interface(s) and the provision of high level local interactivity within the client workstation?

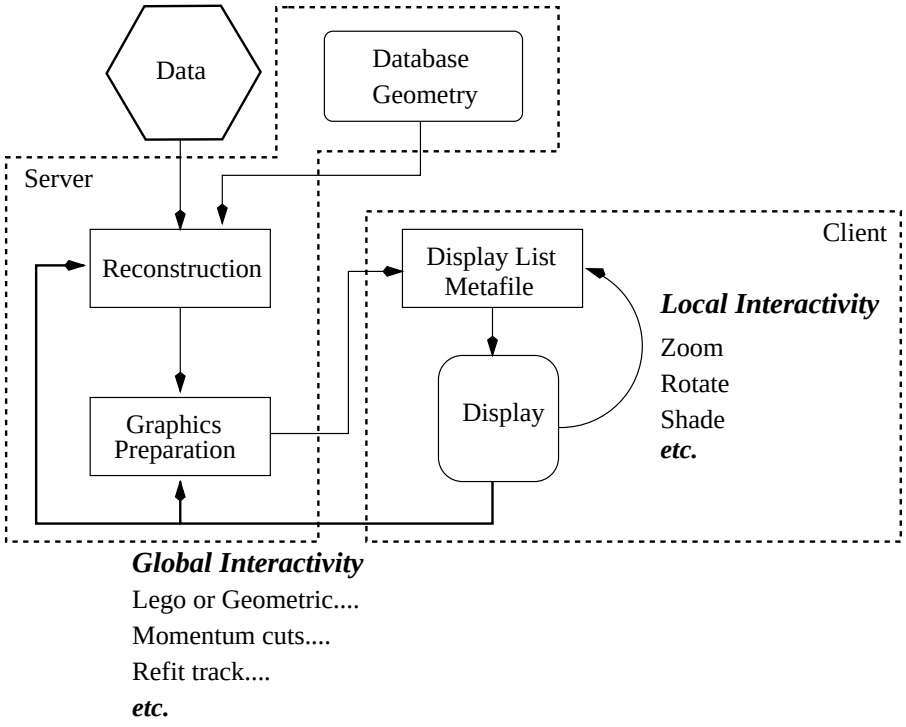

Figure 2: The client-server model of local and global interactity.

- What are the overhead issues associated with data transmission from server to client and how do they relate to the processing capabilities of the client workstation?
- What are the problems associated with processing user commands issued at the client that must direct the processing and flow of data through the tasks running on the server. This "global interactivity" issue is extremely difficult and has only been addressed with limited success in present day visualization systems.

Despite the preceeding discussion the distinction between local and global tasks is not always black and white. Take, for example, the display of a calorimeter "hit" in the detector. These hits may be displayed as objects that correctly represent the geometry of the detector element that was hit and rendered in a color that represents the energy, timing, or some other characteristic, associated with the hit. Alternatively the hit may be displayed as a distorted representation of the detector element where the distortion is used to convey information about the amount of energy associated with the hit, and again color is used to represent other information. In the two cases the actual (x, y, z) values and the type of graphics primitives employed will be different, even though the basic information required to represent the information - at the very least, a unique software identifier for the hit detector element and a word with the encoded energy and timing information - remains the same. It is possible to construct a new graphics primitive, called a supermarker, consisting of the software identifier and some additional information[1]. This is then stored as a graphics object and different representations of this object can then be written to the display list without the requirement of a fully global function. In the future it might be hoped that such functionallity could be coded in a fully local fashion, either by clever use of the display list itself, or (in an ideal case) as a fully supported graphics primitive in some future (HEP friendly) rendering package.

[1] This technique has been extensively exploited by the L3 group.

5 Future Display Systems for HEP

Future event display systems will include all the functionality of today's systems, but will have to deal with much more complex data. This presents a major challenge to their authors who must invent new ways of presenting complex data with abstract representations. Future display systems are likely to be more closely linked to the rest of the experimental software base and probably will incorporate many of the analysis tools that today are in seperate packages (for example, PAW). Given the investment in resources required to produce the user interface of a large visualization system, it is reasonable that the online documentation system of the experiment be incorporated into the same software. Maintaining an up-to-date documentation base is crucial if users are to correctly interpret the data being displayed by the visualization system. The general trend is towards more distributed systems with access over the network and the adoption of commercial standards for user interfaces and graphics. These trends will require a complete understanding of the client/server, local/global models of this software.

6 Conclusions

Visualization software is an important part of a physicist's toolbox at all stages of an experiment and it is important that an adequate amount of the overall resources of the experiment be devoted to the development and maintenance of this software. Often visualization software provides the physicist with his only intuitive window on what happened in a particular event and what is *really* going on in the physics analysis. Visualization provides the surest way of debugging, otherwise absurdly complex, software systems and the fastest way of determining when things are going awray - which can translate to real money when taken in the context of online monitoring of a large experiment. Finally, event visualization tools often provide the only means of communication between the complex and abstract world of particle physics and the general public; their use in education at all levels is not to be undervalued.

Acknowledgements

I would like to acknowledge many useful discussions with Rene Wilhelm, Grassiano Massaro, Sunanda Bannerjee and Francis Bruyant during the design and implimentation of the L3 event visualization system. The (sometimes helpful) feedback from my collaborators on L3 is gratefully acknowledged. I wish to thank the organisers of HEPVIS'96 for a pleasant and stimulating workshop.

References

1 H. Breuker, H. Drevermann, C. Grab, A. Rademakers and H. Stone *Tracking and Imaging Elementary Particles*, Sci. Am. 265 (1991) No. 2 47.
Neil Calder, *HEP Visualisation and the media*, these proceedings.

2 D. McNally, *Event Display at LEP*, these proceedings.

3 H. Drevermann and C. Grab, "Graphical Concepts For The Representation Of Events in High Energy Physics", Int. J. Mod. Phys. C, **1**, pp. 147– 163 (1990).